Probabilistic Event Dropping for Intermittently Connected Subscribers over Pub/Sub Systems

Georgios Bouloukakis*‡, Ioannis Moscholios† Nikolaos Georgantas‡
gboulouk@ics.uci.edu,idm@uop.gr,nikolaos.georgantas@inria.fr
*Donald Bren School of Information and Computer Sciences University of California, Irvine, USA.
†Dept. of Informatics & Telecommunications, University of Peloponnese, Tripolis, Greece.
‡MiMove Team, Inria Paris, France.

***Abstract*—Internet of Things (IoT) aim to leverage data from multiple sensors, actuators and devices for improving peoples' daily life and safety. Multiple data sources must be integrated, analyzed from the corresponding application and notify interested stakeholders. To support the data exchange between data sources and stakeholders, the publish/subscribe (pub/sub) middleware is often employed. Pub/sub provides additional mechanisms such as reliable messaging, event dropping, prioritization, etc. The event dropping mechanism is often used to satisfy Quality of Service (QoS) requirements and ensure system stability. To enable event dropping, basic approaches apply finite buffers or data validity periods and more sophisticated ones are information-aware. In this paper, we introduce a pub/sub mechanism for probabilistic event dropping by considering the stakeholders' intermittent connectivity and QoS requirements. We model the pub/sub middleware as a network of queues which includes a novel ON/OFF queueing model that enables the definition of join probabilities. We validate our analytical model via simulation and compare our mechanism with existing ones. Experimental results can be used as insights for developing hybrid dropping mechanisms.**

***Index Terms*—Publish/Subscribe Middleware, Event Dropping, Response Time, Queueing Networks**

I. Introduction

Internet of Things (IoT) devices are prevalent in homes, offices, and community spaces such as universities, hospitals, airports, etc [1]. By exploiting the information they gather there are new opportunities to build applications that improve peoples' quality of life. However, IoT devices are often mobile, low-powered and inexpensive which makes them vulnerable to system changes. These changes may occur due to a variety of problems including faulty components, inaccurate sensing, intermittent connectivity and software bugs [2].

Such a system behavior prevents the development of resilient IoT applications, especially those that carry mission-critical information to improve public safety [3], [4]. For example, during a fire inside a building an emergency dispatch process must be activated. This may result to sending a team of Fire Fighters (FFs) to the building. Data (e.g., high temperature or smoke levels) derived from the building's IoT devices along with static information (e.g, building floor plans) can be leveraged by stakeholders (e.g., FFs, residents) to ensure their safety. However, the key challenge for this unreliable, partially available and congested network environment is to deliver data in a timely manner [5].

To manage data flows of IoT applications, an underlying data exchange middleware can be leveraged [6]. Middleware systems often follow the Publish/Subscribe (pub/sub) interaction paradigm, which supports space and time decoupling between peers [7]. This is particularly needed in the case of mobile IoT systems, where intermittent connected peers (e.g., FFs) interact with each other via an intermediary broker [8]. Pub/sub middleware systems provide multiple features to support data exchange such as reliable messaging, message prioritization, load balancing, message dropping, and others [9]–[12]. These features can be utilized to ensure a certain level of Quality of Service (QoS) for IoT applications. For instance, IoT applications evolve over time: publication rates increase, the network may be congested, the bandwidth may become limited, and peers often connect/disconnect from the network. Hence, to enable the broker's buffer stability and prevent high delays, the feature of message dropping is highly useful.

Several existing efforts support message dropping. In particular, message brokers such as RabbitMQ [8], ActiveMQ [13], mosquitto [14], set a maximum queue length and drop messages that arrive when the queue is full [12]. To ensure message timeliness, message losses occur due to the validity/availability periods that can be assigned to every message through pub/sub protocols and APIs (e.g., the JMS API [11]). More sophisticated approaches support semi-probabilistic delivery [15] or take into account subscribers' preferences [10], [16], [17] to: *(i)* satisfy subscribers based on the available bandwidth usage; *(ii)* deliver the most relevant publications published within a time window; or *(iii)* deliver most popular publications. In our recent work, we introduced FireDeX [18], a middleware architecture that enables message prioritization and dropping by considering subscribers' situational awareness requirements and resource constraints.

In this paper, we introduce a pub/sub architecture enabled with probabilistic message dropping. In particular, our pub/sub broker drops messages by considering resource constraints, the subscribers' connectivity and response time requirements. We model the pub/sub broker as a network of queues where messages are transmitted to each subscriber through a dedicated ON/OFF probabilistic queueing model – ON/OFF corre-

TABLE I: Model variables' and shorthand notation.

Variable(s)	Definition/Description
$v_k \in V$, $r_j \in R$	event topics and subscriptions
$p_i \in P$, $s_i \in S$	publishers and subscribers
$\lambda^{pub}_{p_i,v_k}$, λ^{in}_b, λ^{nosub}	publication rate, input b's rate, subscriptions drop rate
$\lambda^{notify}_{s_i}$, $\lambda^{sub}_{s_i}$	s_i's notification rate, s_i's delivery rate
ζ_{s_i}, S_b	join probability, $\|S\|$ subscribed in b
μ^{in}, $\mu^{out}_{s_i}$	processing rate, transmission rate
$T^{s_i}_{\text{ON}}$, $T^{s_i}_{\text{OFF}}$	average connected period, average disconnected period
Δ_{s_i}, $\Delta^{Thr}_{s_i}$, Ξ_{s_i}	s_i's response time, s_i's threshold, s_i's success rate

sponds to the subscriber's connected/disconnected states. An analytical model estimates the message dropping probability for satisfying specific response times. In our experimental results we validate the analytical model by relying on an open source simulator [19]. Additionally, our approach ensures 100% message success rates during the subscribers' connected periods in comparison to others.

The core contributions of this paper are:

1) A formal model of a pub/sub probabilistic message dropping mechanism during the subscribers' disconnections for satisfying response time requirements (§II).
2) An extensive analysis of the *probabilistic ON/OFF queueing model* with an intermittent available server and probabilistic arrival rates (§III).
3) The validation of the proposed model by relying on an open source simulator (§IV).
4) The comparison of the proposed message dropping mechanism against several others via simulations (§IV).

II. System Model

In this section we provide a formal model of our data exchange middleware architecture that supports probabilistic message dropping. Our middleware builds over the pub/sub interaction paradigm and the probabilistic dropping is performed at the broker component.

A. Pub/Sub Interaction Paradigm

The Pub/Sub interaction paradigm, is commonly used for content broadcasting/feeds. Middleware protocols such as MQTT [20] and AMQP [21], as well as tools and technologies such as RabbitMQ [8] and JMS [11] follow the pub/sub paradigm [7]. In pub/sub, multiple peers interact via an intermediate *broker*. Publishers produce *events* (or messages) characterized by a specific *topic* to the broker. Subscribers subscribe their interest for specific topics to the broker, who maintains an up-to-date list of subscriptions. The broker matches received events with subscriptions and delivers a copy of each event to each interested subscriber.

B. Pub/Sub Formal Model

To formulate the probabilistic pub/sub middleware we rely on our previous experience [18], [22]–[25] where we model pub/sub systems with different characteristics as network of queues. In this work, we do not provide the detailed modeling of our pub/sub middleware (can be found in [22], [24]) but only the added characteristics for enabling probabilistic dropping. Refer to Table I for the notations used throughout this section.

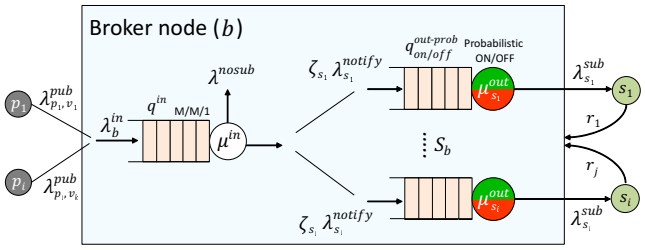

Fig. 1: Pub/Sub formal model.

As depicted in Fig. 1, every publisher p_i publishes events to broker node b at multiple topics (e.g., "smoke") with rate $\lambda^{pub}_{p_i,v_k}$ based on a Poisson process. On the other side, a subscriber s_i subscribes using a subscription r_j to b for receiving events (matching r_j) with rate $\lambda^{sub}_{s_i}$. Published events arrive with rate λ^{in}_b to b's input M/M/1 queue (q^{in}) to be processed with rate μ^{in}. Events not matching b's subscriptions are dropped with rate λ^{nosub} and events matching s_i's subscriptions are forwarded to s_i's output queue ($q^{out-prob}_{on/off}$) with rate $\lambda^{notify}_{s_i}$.

Subsequently, events join s_i's output queue with probability ζ_{s_i} only when s_i is not connected to b. Hence, the delivery rate of events for each subscriber s_i can be estimated by taking into account s_i's join probability ζ_{s_i} and its matched events with rate $\lambda^{notify}_{s_i}$. Finally, events inserted to the output queue during s_i's disconnection periods are buffered because the queueing server is not active. When s_i re-connects to the broker, events are served with rate $\mu^{out}_{s_i}$ which represents the network transmission delay between b and s_i. Let Δ_{s_i} be the *average end-to-end response time* of events matching s_i's subscriptions from the moment they are published until s_i receives them. To estimate Δ_{s_i}, we must estimate the delay of events passing through the broker's input and output queues. This is given by:

$$\Delta_{s_i} = \Delta_{q^{in}} + \Delta_{q^{prob}_{on/off}} \quad (1)$$

where $\Delta_{q^{in}}$ and $\Delta_{q^{prob}_{on/off}}$ are the mean response times for the queues $q^{prob}_{on/off}$ and q^{in}, respectively. To estimate $\Delta_{q^{in}}$ we leverage existing solutions of the literature [26]. However, $\Delta_{q^{prob}_{on/off}}$ cannot be estimated in a similar way; we must take into account the subscribers intermittent connectivity, the join probability during disconnections, as well as the network transmission delay between b and s_i.

C. Probabilistic Event Dropping

In this paper, our pub/sub broker estimates the join probability ζ_{s_i} per subscriber s_i, by taking into account the subscriber's connected and disconnected periods $T^{s_i}_{\text{ON}}$ and $T^{s_i}_{\text{OFF}}$, and its end-to-end response time threshold $\Delta^{Thr}_{s_i}$. The latter requires that all events matching s_i's subscriptions must be delivered within $\Delta^{Thr}_{s_i}$. This threshold is given to the broker through a subscription which is defined as a tuple $r_j = (s_i, \Delta^{Thr}_{s_i})$. Note that $\Delta^{Thr}_{s_i}$ is constant for any r_j of s_i and it is equivalent to the end-to-end response time Δ_{s_i} defined in (1). Additionally,

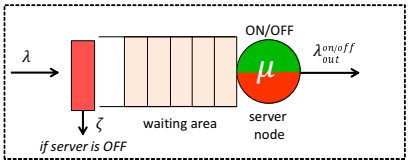

Fig. 2: Probabilistic ON/OFF queue.

$T_{\text{ON}}^{s_i}$ and $T_{\text{OFF}}^{s_i}$ periods are estimated statistically inside b based on s_i's ON/OFF states by relying on a monitoring tool.

The above behavior can be modeled as a *probabilistic ON/OFF queue* (output queue inside the broker – see Fig. 1), which we will analyze in the next section to derive its performance metrics. Note that our probabilistic mechanism drops all incoming events during disconnections – the event's importance is not considered. Such information awareness is handled in our previous work [18].

III. Probabilistic ON/OFF model

An ON/OFF queue represents a system having a single *server* as depicted in Fig. 2. The server supports an exponentially distributed service rate denoted as $\mu > 0$ (time needed to process an event). The service station operates under the First-Come-First-Serve (FCFS) queueing policy. We also assume that the server is subject to an ON/OFF procedure. That said, it remains in the ON-state for an exponentially distributed time with parameter θ_{ON}, during which it serves events, if any. Let T_{ON} be the time the server is ON – then $T_{\text{ON}} = 1/\theta_{\text{ON}}$. When this time expires, the server enters the OFF-state during which it stops working for an exponentially distributed time period with rate θ_{OFF}. Let T_{OFF} be the time the server is OFF – then $T_{\text{OFF}} = 1/\theta_{\text{OFF}}$.

Events arrive in the system according to a Poisson process with rate $\lambda > 0$ and if the server is online (ON) then events are placed in a queue waiting to be "served". Otherwise, if the server is offline (OFF) then events join the queue with probability ζ or abandon the system with probability $1 - \zeta$.

To model the performance of a component that sends events according to the above connectivity, we introduce the "probabilistic ON/OFF queue". A $q_{on/off}^{prob}$ queue is defined as:

$$q_{on/off}^{prob} = (\lambda, \zeta, \mu, \lambda_{out}^{on/off}, T_{\text{ON}}, T_{\text{OFF}}) \tag{2}$$

where λ is the input rate of events to the queue, ζ is the probability of joining the queue, $\lambda_{out}^{on/off}$ is the output rate of events and μ is the service rate for the transmission of events, if any, during T_{ON}. The output process is intermittent, since no events exit the queue during T_{OFF} intervals. Without loss of generality, we assume that: if T_{ON} expires and there is an in-service event, the server interrupts its processing and then continues in the next T_{ON} period.

It is worth noting that if $\zeta = 1$, then all events join the system during OFF periods. This is equivalent with the ON/OFF model presented in [23]. Hence, the probabilistic ON/OFF model generalizes the ON/OFF model.

A. The analytical model

The proposed model is described as a 2D Markov chain whose state space diagram is presented in Fig. 3. The states $(n,1)$ and $(n,0)$ refer to the case where there are n events in the system (queue + server) and the server is ON and OFF, respectively. Note that, if the server is always ON then we have an M/M/1 queue and the upper part of the 2D Markov chain does not exist. Based on this chain, the steady state probabilities $P_{n,k}$ (n=0,1,... while k=0,1) of the ON/OFF queue do not have a product-form solution since there are no backward transitions between the states $(n,0)$ and $(n\text{-}1, 0)$. This is anticipated since events are not serviced when the server is OFF.

According to Fig. 3, we can write:

$$\theta_{\text{OFF}} \sum_{n=0}^{\infty} P_{n,0} = \theta_{\text{ON}} \sum_{n=0}^{\infty} P_{n,1} \tag{3}$$

Equation (3), based on the normalization condition $\sum_{n=0}^{\infty}\sum_{k=0}^{1} P_{n,k} = 1$, leads to the determination of the probability of when the server is ON (see (4)) or OFF (see (5)), respectively:

$$P_{server}(ON) = \sum_{n=0}^{\infty} P_{n,1} = \frac{\theta_{\text{OFF}}}{\theta_{\text{ON}} + \theta_{\text{OFF}}} \tag{4}$$

$$P_{server}(OFF) = \sum_{n=0}^{\infty} P_{n,0} = \frac{\theta_{\text{ON}}}{\theta_{\text{ON}} + \theta_{\text{OFF}}} \tag{5}$$

The global balance (GB) equations of the 2D Markov chain are (rate into state (n,k) - rate out of state (n, k) = 0):

State(0,0) : $\theta_{\text{ON}} P_{0,1} - (\lambda\zeta + \theta_{\text{OFF}}) P_{0,0} = 0$

State(0,1) : $\theta_{\text{OFF}} P_{0,0} + \mu P_{1,1} - (\lambda + \theta_{\text{ON}}) P_{0,1} = 0$

State(1,0) : $\lambda\zeta P_{0,0} + \theta_{\text{ON}} P_{1,1} - (\lambda\zeta + \theta_{\text{OFF}}) P_{1,0} = 0$

State(1,1) : $\lambda P_{0,1} + \mu P_{2,1} + \theta_{\text{OFF}} P_{1,0} - (\lambda + \mu + \theta_{\text{ON}}) P_{1,1} = 0$

...

State(n,0) : $\lambda\zeta P_{n-1,0} + \theta_{\text{ON}} P_{n,1} - (\lambda\zeta + \theta_{\text{OFF}}) P_{n,0} = 0$

State(n,1) : $\lambda P_{n-1,1} + \mu P_{n+1,1} + \theta_{\text{OFF}} P_{n,0} - (\lambda + \mu + \theta_{\text{ON}}) P_{n,1} = 0$

...

The transition (intensity) matrix, $\boldsymbol{Q}$, of this 2D Markov chain is an infinite block tridiagonal matrix whose structure is repetitive. More precisely, $\boldsymbol{Q}$ can be written in the form:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{B_{00}} & \boldsymbol{C_0} & \boldsymbol{0} & \boldsymbol{0} & \dots \\ \boldsymbol{C_2} & \boldsymbol{C_1} & \boldsymbol{C_0} & \boldsymbol{0} & \dots \\ \boldsymbol{0} & \boldsymbol{C_2} & \boldsymbol{C_1} & \boldsymbol{C_0} & \dots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{C_2} & \boldsymbol{C_1} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{6}$$

where submatrices $\boldsymbol{B_{00}}$, $\boldsymbol{C_0}$, $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ are square 2x2 matrices with the following form:

$$\boldsymbol{B_{00}} = \begin{bmatrix} -(\lambda\zeta + \theta_{\text{OFF}}) & \theta_{\text{OFF}} \\ \theta_{\text{ON}} & -(\lambda + \theta_{\text{ON}}) \end{bmatrix} \tag{7}$$

$$\boldsymbol{C_0} = \begin{bmatrix} \lambda\zeta & 0 \\ 0 & \lambda \end{bmatrix} \tag{8}$$

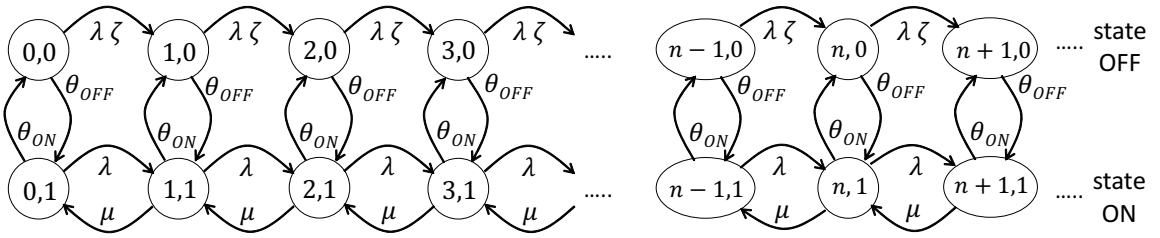

Fig. 3: 2D Markov Chain.

$$\boldsymbol{C_1} = \begin{bmatrix} -(\lambda\zeta + \theta_{\mathrm{OFF}}) & \theta_{\mathrm{OFF}} \\ \theta_{\mathrm{ON}} & -(\lambda + \mu + \theta_{\mathrm{ON}}) \end{bmatrix} \tag{9}$$

$$\boldsymbol{C_2} = \begin{bmatrix} 0 & 0 \\ 0 & \mu \end{bmatrix} \tag{10}$$

A Markov chain whose transition matrix has the structure of (6) belongs to the class of quasi-birth death processes and can be efficiently solved using the matrix-geometric method [27]. Based on (6), the system of GB equations is described by:

$$\boldsymbol{PQ} = \boldsymbol{0} \tag{11}$$

where $\boldsymbol{P} = [\boldsymbol{P_0}, \boldsymbol{P_1}, ..., \boldsymbol{P_n}, ...]$, $\boldsymbol{P_n} = [P_{n,0}, P_{n,1}]$ and $P_{n,k}$ is the probability that the system is in state (n, k) at steady state.

Based on (11) we can write the following equations:

$$\boldsymbol{P_0 B_{00}} + \boldsymbol{P_1 C_2} = \boldsymbol{0}, \quad \text{for } \boldsymbol{n = 0} \tag{12a}$$

$$\boldsymbol{P_{n-1} C_0} + \boldsymbol{P_n C_1} + \boldsymbol{P_{n+1} C_2} = \boldsymbol{0}, \quad \text{for } \boldsymbol{n \geq 1} \tag{12b}$$

The steady-state probability distribution, $\boldsymbol{P_n}$, is matrix-geometric in form, i.e.:

$$\boldsymbol{P_n} = \boldsymbol{P_0 R^n} \quad \text{for } n = 1, 2, ..., \tag{13}$$

where $\boldsymbol{R}$ is a constant rate matrix.

Thus, the GB equations (12a) and (12b) can be written via (13) as follows:

$$\boldsymbol{P_0 B_{00}} + \boldsymbol{P_0 R C_2} = \boldsymbol{0}, \quad \text{for } \boldsymbol{n = 0} \tag{14a}$$

$$\boldsymbol{P_0 R^{n-1} C_0} + \boldsymbol{P_0 R^n C_1} + \boldsymbol{P_0 R^{n+1} C_2} = \boldsymbol{0} \quad \textit{or}$$

$$\boldsymbol{P_0 R^{n-1}}\big[\boldsymbol{C_0} + \boldsymbol{R C_1} + \boldsymbol{R^2 C_2}\big] = \boldsymbol{0}, \quad \text{for } \boldsymbol{n \geq 1} \tag{14b}$$

To determine the rate matrix $\boldsymbol{R}$ we follow the logarithmic reduction process proposed in [28]. Latouce and Ramaswami introduce a rate matrix $\boldsymbol{G}$ that satisfies:

$$\boldsymbol{C_2} + \boldsymbol{C_1 G} + \boldsymbol{C_0 G^2} = \boldsymbol{0} \tag{15}$$

Having determined $\boldsymbol{G}$ we can compute $\boldsymbol{R}$ via the formula:

$$\boldsymbol{R} = -\boldsymbol{C_0}(\boldsymbol{C_1} + \boldsymbol{C_0 G})^{-1} \tag{16}$$

In our case, we may express $\boldsymbol{C_2}$ via a column vector $\boldsymbol{v}$ and a row vector $\boldsymbol{u}$ as follows:

$$\boldsymbol{C_2} = \begin{bmatrix} 0 & 0 \\ 0 & \mu \end{bmatrix} = \begin{bmatrix} 0 \\ \mu \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \boldsymbol{v \cdot u} \tag{17}$$

Then:

$$\boldsymbol{G} = \boldsymbol{1 \cdot u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \tag{18}$$

Equation (16) based on (18) gives:

$$\boldsymbol{R} = p \begin{bmatrix} \frac{\zeta(\mu + \theta_{\mathrm{ON}})}{\lambda\zeta + \theta_{\mathrm{OFF}}} & \zeta \\ \frac{\theta_{\mathrm{ON}}}{\lambda\zeta + \theta_{\mathrm{OFF}}} & 1 \end{bmatrix} \tag{19}$$

where p=λ/μ.

Having determined $\boldsymbol{R}$, we can compute $\boldsymbol{P_0}$. Based on [27] (Theorem 1.2.1, Chapter 1), we have that the vector $\boldsymbol{P_0}$ should simultaneously satisfy:

$$\boldsymbol{P_0}(\boldsymbol{B_{00}} + \boldsymbol{R C_2}) = \boldsymbol{0} \tag{20}$$

$$\boldsymbol{P_0}(\boldsymbol{I} - \boldsymbol{R})^{-1}\boldsymbol{e} = 1 \tag{21}$$

where $\boldsymbol{e}$ refers to the eigenvalues of $\boldsymbol{R}$. The solution of (20), (21) results in:

$$\begin{aligned} \boldsymbol{P_0} &= \begin{bmatrix} P_{0,0} & P_{0,1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\theta_{\mathrm{ON}}}{(\lambda\zeta + \theta_{\mathrm{OFF}})} \left(\frac{\theta_{\mathrm{OFF}}(1-p) - p\zeta\theta_{\mathrm{ON}}}{\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}} \right) & \frac{\theta_{\mathrm{OFF}}(1-p) - p\zeta\theta_{\mathrm{ON}}}{\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}} \end{bmatrix} \end{aligned} \tag{22}$$

Based on (13) and (22), we can compute $\boldsymbol{P_n}$ for $\boldsymbol{n}$=1,2,... From $\boldsymbol{P_n}$ we calculate the average number of events in the system (server + queue), $E(n)_{on/off}$ via:

$$E(n)_{on/off} = \sum_{n=0}^{\infty} n\big(P_{n,0} + P_{n,1}\big) \tag{23}$$

Since:

$$P_{n,0} + P_{n,1} = \boldsymbol{P_n} \cdot \boldsymbol{1} = \boldsymbol{P_0 R^n} \cdot \boldsymbol{1} \tag{24}$$

we can write (23) as follows:

$$\begin{aligned} E(n)_{on/off} = \sum_{n=0}^{\infty} n\boldsymbol{P_0 R^n} \cdot \boldsymbol{1} = \boldsymbol{P_0 R} \sum_{n=0}^{\infty} n\boldsymbol{R^{n-1}} \cdot \boldsymbol{1} = \\ \boldsymbol{P_0 R}\Big[(\boldsymbol{I} - \boldsymbol{R})^{-1}\Big]^{\boldsymbol{2}} \cdot \boldsymbol{1} = A_1(A_2 + A_3) \end{aligned} \tag{25}$$

where:

$$A_1 = \frac{p}{\big(\theta_{\mathrm{OFF}}(1-p) - p\zeta\theta_{\mathrm{ON}}\big)\big(\lambda\zeta + \theta_{\mathrm{OFF}}\big)\big(\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}\big)}$$

$$A_2 = \theta_{\mathrm{ON}}\Big(\zeta(\lambda + \mu + \theta_{\mathrm{ON}}) + \theta_{\mathrm{OFF}}\Big)\Big((\lambda\zeta + \theta_{\mathrm{OFF}})(1-p)(1-p+p\zeta) + p\zeta\big(p\theta_{\mathrm{ON}}(1-\zeta) + \theta_{\mathrm{OFF}}\big)\Big)$$

$$A_3 = \Big(\zeta(\lambda + \theta_{\mathrm{ON}}) + \theta_{\mathrm{OFF}}\Big)\Big(p\theta_{\mathrm{ON}}(\lambda\zeta + \theta_{\mathrm{OFF}})(1-p+p\zeta) + (\theta_{\mathrm{OFF}} - p\zeta\theta_{\mathrm{ON}})\big(p\theta_{\mathrm{ON}}(1-\zeta) + \theta_{\mathrm{OFF}}\big)\Big)$$

If ζ=1, i.e., if events always join the queue then (22) becomes:

$$\begin{aligned} \boldsymbol{P_0} &= \begin{bmatrix} P_{0,0} & P_{0,1} \end{bmatrix} = \\ &\begin{bmatrix} \frac{\theta_{\mathrm{ON}}}{(\lambda + \theta_{\mathrm{OFF}})} \left(\frac{\theta_{\mathrm{OFF}}}{\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}} - p \right) & \frac{\theta_{\mathrm{OFF}}}{\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}} - p \end{bmatrix} \end{aligned} \tag{26}$$

Similarly, (25) becomes [23]:

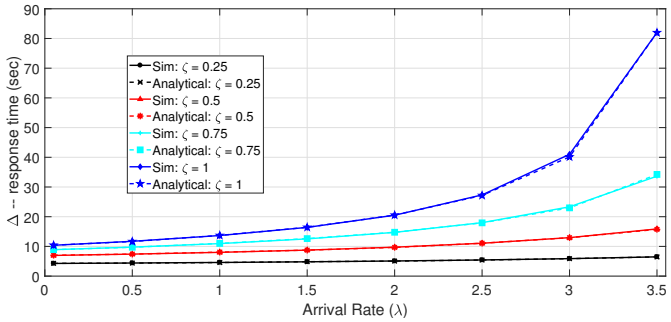

Fig. 4: Probabilistic ON/OFF model validation.

$$E(n)_{on/off} = \frac{\lambda\theta_{\mathrm{ON}} + p\big(\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}\big)^2}{\Big(\theta_{\mathrm{OFF}} - p\big(\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}\big)\Big)\Big(\theta_{\mathrm{ON}} + \theta_{\mathrm{OFF}}\Big)} = \frac{p^{'} + \lambda T_{\mathrm{OFF}} P_{server}(OFF)}{1 - p^{'}} \tag{27}$$

where $p^{'} = \lambda/\mu \cdot P_{server}\big(ON\big)$.

Having determined $E(n)_{on/off}$ via (23) (or via (27), assuming ζ=1), we can calculate the average system time (server + queue), $\Delta_{q^{prob}_{on/off}}$, via Littles law as follows:

$$\Delta_{q^{prob}_{on/off}} = \frac{E(n)_{on/off}}{\lambda_{eff}} \tag{28}$$

where $\lambda_{eff} = \lambda\ \big(1\text{-}P_{server}(OFF)(1\text{-}\zeta)\big) = \lambda^{on/off}_{out}$. We leverage (28) to estimate (1) where $\lambda = \lambda^{notify}_{s_i}$ and $\mu = \mu^{out}_{s_i}$.

IV. Experimental Results

We now validate the probabilistic ON/OFF model and evaluate the event dropping mechanism of our approach. For this, we use the JINQS (Java simulation library for multiclass queueing networks) open source simulator which enables system designers to build simulations for a wide range of queueing networks [19]. We extend JINQS to: *(i)* implement the probabilistic ON/OFF model; and *(ii)* simulate the proposed event-dropping mechanism as well as other existing approaches of pub/sub systems [11], [12].

A. Analytical vs. simulated response time

After implementing the probabilistic ON/OFF queue using our simulator, we parameterize it as follows: *(i)* the server remains in the ON and OFF states for exponentially distributed time periods $T_{\mathrm{ON}} = T_{\mathrm{OFF}} = 20$ sec, thus, the server processes events and then buffers them every 20 sec on average; *(ii)* during the server's OFF periods, events join the queue with probability ζ which takes on values: 0.25, 0.5, 0.75 and 1.0; *(iii)* events are processed with a mean service rate μ = 8 events/sec; *(iv)* there is sufficient buffer capacity so that no events are dropped because of this; and *(v)* events arrive to the queue with a mean rate varying from 0.05 to 3.5 events/sec. We apply the above parameters to our simulation model and plot curves of mean response time (server+queue) as depicted in Fig. 4. The analytical results obtained by (28) and depicted also in Fig. 4, show the high accuracy of (28). Note that when applying a join probability of ζ = 1, the ON/OFF queueing system is saturated for arrival rates greater than 4 events/sec. For $\zeta < 1$, the system drops events and thus it is possible to apply arrival rates greater than 4 events/sec based on the condition 1 - $\big(\lambda_{eff}/\mu\big) < 1$.

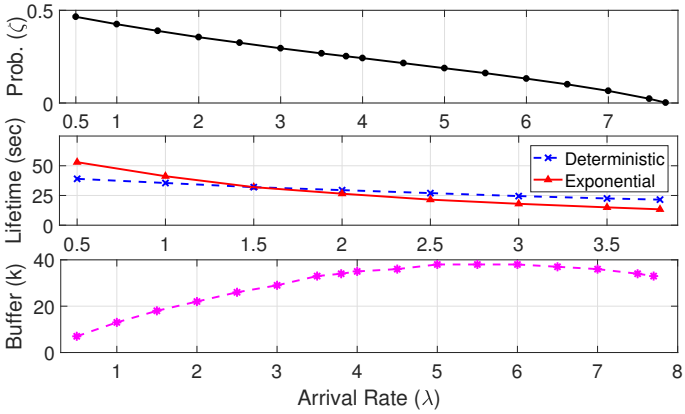

Fig. 5: Applying parameters for comparison.

B. Event dropping comparison

We now compare the proposed event dropping mechanism against other mechanisms [11], [12] found in the literature of pub/sub systems. We set up our system using our simulator and the queueing network of Fig. 1. Events arrive at the input queue (q^{in}) to be processed with rate μ^{in} = 16 events/sec. There is no event dropping because of subscriptions, thus events arrive at the subscriber's ON/OFF queue to be transmitted with rate $\mu^{out}_{s_i}$ = 8 events/sec. The server of the ON/OFF queue changes its ON/OFF state based on the subscribers intermittent connectivity – i.e., $T^{s_i}_{\mathrm{ON}} = T^{s_i}_{\mathrm{OFF}} = 20$ sec.

Upon s_i's subscription (r_j), the response time threshold $\Delta^{Thr}_{s_i}$ = 7 sec is introduced and so the subscriber must receive all events within 7 sec on average. Hence, the pub/sub system must drop events to achieve this. We select this threshold because $T^{s_i}_{\mathrm{ON}} = T^{s_i}_{\mathrm{OFF}} = 20$ sec, and thus for any arrival rate (λ^{in}_b) or processing rates (μ^{in}, $\mu^{out}_{s_i}$), the end-to-end response time cannot be less than 10 sec without event dropping. By setting $\Delta^{Thr}_{s_i}$ = 7 sec, we enforce event dropping even for very low λ^{in}_b. We compare the following dropping mechanisms:

1) *Probabilistic event dropping (proposed)*: events join the queue with a join probability ζ, which is estimated based on $\Delta^{Thr}_{s_i}$ = 7 sec and (28).

2) *Expired event dropping* [11]: upon the publication of an event, a *lifetime* period is applied to represent the event validity inside the queueing network. Hence, an event may enter the queuing network when it is published, but will leave the network as soon as its lifetime elapses and the event is considered expired. This corresponds to an ON/OFF queue with reneging or impatient customers.

3) *Overloaded event dropping* [12]: we apply finite capacity buffers K at both input and output queues $\big(q^{in}, q^{out-prob}_{on/off}\big)$. Events that overload the queue will be dropped.

We run each experiment for a different arrival rate λ^{in}_b: 0.05 - 7.7 events/sec. We apply the parameters depicted in Fig. 5 depending on the applied mechanisms. For the *expired event*

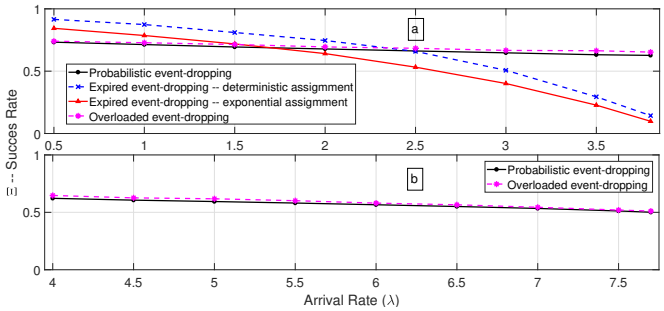

Fig. 6: Event-dropping mechanism comparisons.

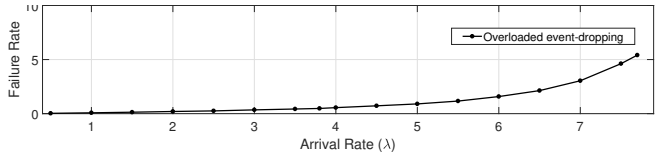

Fig. 7: Failure rates during ON periods.

dropping (second) experiment, we apply either deterministic *lifetime* for each λ_b^{in} or based on an exponential distribution. Note that for the second and third mechanisms, the join probability is set to $\zeta = 1$ – i.e., all events join the $q_{on/off}^{out-prob}$ queue.

Fig. 6 shows the delivery success rates of all three dropping mechanisms for $\Delta_{s_i} = 7$ sec $= \Delta_{s_i}^{Thr}$. For $\lambda_b^{in} < 2$ the success rates are greater than 67% for both the deterministic and exponential expired dropping mechanism. It is worth noting that the expired event dropping mechanism provides the highest delivery success rates in comparison to the remaining two. However, for $\lambda_b^{in} > 2$, success rates decrease suddenly. Regarding the other two mechanisms, the success rate of the overloaded event dropping is slightly higher ($\sim$1-2% greater) than the probabilistic event dropping for $\lambda_b^{in} > 0.05$ and $\lambda_b^{in} < 7.7$.

On the other hand, Fig. 7 shows the failure rate of events during s_i's connection (ON) periods when applying the overloaded event dropping mechanism – in the probabilistic mechanism there are no failures during ON periods, we only drop events during OFF periods. Therefore, even if the subscriber is connected, events may be dropped with rate 6% for higher arrival rates.

V. Conclusion

IoT devices usually exchange data by relying on the pub/sub middleware. Among the different mechanisms, pub/sub often supports event dropping for satisfying subscribers' QoS requirements. In this paper, we introduce a different event dropping mechanism that takes into account the subscribers' QoS semantics and requirements. We model the performance of a pub/sub middleware employing this mechanism and solve it analytically using queueing theory. We validate our analytical model using simulations and we compare our mechanisms against others in pub/sub systems. Our mechanism provides the highest delivery success rate when applying high publication rates and subscribers are connected.

In our future work, we intend to implement a prototype of the proposed mechanism.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, pp. 22–32, 2014.

[2] P. H. Su, C.-S. Shih, J. Y.-J. Hsu, K.-J. Lin, and Y.-C. Wang, "Decentralized fault tolerance mechanism for intelligent iot/m2m middleware," in *IEEE WF-IoT*, 2014.

[3] "Community Seismic Network." http://csn.caltech.edu, May 2015.

[4] E. Cochran, J. Lawrence, C. Christensen, and A. Chung, "A novel strong-motion seismic network for community participation in earthquake monitoring," *IEEE Instrumentation & Measurement Magazine*, 2009.

[5] A. Hamins, C. Grant, N. Bryner, A. Jones, and G. Koepke, *NIST Special Publication 1191 Research Roadmap for Smart Fire Fighting*. National Institute Of Standards and Technology, 2015.

[6] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, pp. 1–20, 2017.

[7] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, 2003.

[8] Pivotal, "RabbitMQ", https://www.rabbitmq.com/, 2018.

[9] C. Esposito, M. Platania, and R. Beraldi, "Reliable and timely event notification for publish/subscribe services over the internet," *IEEE/ACM Transactions on Networking*, 2014.

[10] M. Diallo, S. Fdida, V. Sourlas, P. Flegkas, and L. Tassiulas, "Leveraging caching for internet-scale content-based publish/subscribe networks," in *IEEE ICC*, 2011.

[11] Sun Microsystems JMS Specifications, http://www.oracle.com/technetwork/java/jms/index.html, 2018.

[12] V. John and X. Liu, "A survey of distributed message broker queues," *arXiv preprint arXiv:1704.00411*, 2017.

[13] B. Snyder, D. Bosnanac, and R. Davies, *ActiveMQ in action*, 2011.

[14] R. Light, "Mosquitto-an open source mqtt v3. 1 broker," *URL: http://mosquitto. org*, 2013.

[15] P. Costa, G. P. Picco, and S. Rossetto, "Publish-subscribe on sensor networks: A semi-probabilistic approach," in *IEEE MASS*, 2005.

[16] K. Pripužić, I. P. Žarko, and K. Aberer, "Top-k/w publish/subscribe: finding k most relevant publications in sliding time window w," in *ACM DEBS*, 2008.

[17] P. Salehi, K. Zhang, and H.-A. Jacobsen, "Popsub: Improving resource utilization in distributed content-based publish/subscribe systems," in *ACM DEBS*, 2017.

[18] K. E. Benson, G. Bouloukakis, C. Grant, V. Issarny, S. Mehrotra, I. Moscholios, and N. Venkatasubramanian, "Firedex: a prioritized iot data exchange middleware for emergency response," *ACM/IFIP/USENIX International Middleware Conference*, Rennes, France, Dec. 2018.

[19] T. Field, "Jinqs: An extensible library for simulating multiclass queueing networks, v1. 0 user guide," Aug. 2006.

[20] A. Banks and R. Gupta, "Mqtt version 3.1. 1," *OASIS standard*, 2014.

[21] R. Godfrey, D. Ingham, and R. Schloming, "Oasis advanced message queuing protocol (amqp) version 1.0; oasis standard," 2012.

[22] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny, "Timeliness evaluation of intermittent mobile connectivity over pub/sub systems," in *ACM/SPEC ICPE*, L'Aquila, Italy, Apr. 2017.

[23] G. Bouloukakis, I. Moscholios, N. Georgantas, and V. Issarny, "Performance modeling of the middleware overlay infrastructure of mobile things," in *IEEE ICC*, Paris, France, May 2017.

[24] R. Gomes, G. Bouloukakis, F. Costa, N. Georgantas, and R. Da Rocha, "Qos-aware resource allocation for mobile iot pub/sub systems," in *ICIOT*. Springer, Cham, Seattle, USA, 2018.

[25] G. Bouloukakis, A. Kattepur, N. Georgantas, and V. Issarny, "Queueing network modeling patterns for reliable and unreliable publish/subscribe protocols," in *MobiQuitous*, New York, USA, 2018.

[26] D. Gross, J. Shortle, J. Thompson, and C. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[27] M. F. Neuts, "Matrix-geometric solutions in stochastic models: an algorithmic approach," *Dover Publications*, 1981.

[28] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. ASA-SIAM series on statistics and applied probability, 1999.